



EBC Debugger User Manual

March 2007

Revision 0.21

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2007, Intel Corporation. All rights reserved.

Contents

1	Introduction	1
1.1	Overview	1
1.2	Terms.....	1
1.3	Conventions used in this document	2
1.3.1	Pseudo-code conventions	2
1.3.2	Typographic conventions	2
1.4	Related Information	3
2	Getting Started.....	5
2.1	What is the EBC Debugger?	5
2.2	Where is EBC Debugger	5
2.3	Prerequisite	5
2.4	Load the EBC Debugger.....	5
2.5	Run the EBC Debugger.....	6
2.6	A typical EBC Debug session	6
3	EBC Debugger Command Description.....	9
3.1	Overview	9
3.1.1	Command Summary	9
3.1.2	Explanation of Command Description Layout	10
3.2	EBC Debugger Commands	11
3.2.1	Execution class commands	11
	G	11
	T	12
	P	13
	O	14
	Q	15
3.2.2	Break class commands.....	16
	BOC	16
	BOCX	17
	BOR	18
	BOE	19
	BOT	20
	BOK	21
	BL	22
	BP	23
	BC	24
	BD	25
	BE	26
3.2.3	Information class commands.....	27
	K	27
	TRACE	28
	R	29
	L	30
	SCOPE	31
	DB, DW, DD, DQ.....	32

	EB, EW, ED, EQ	33
3.2.4	Symbol class commands	34
	LN	34
	LOADSYMBOL	36
	UNLOADSYMBOL	37
	LOADCODE	38
	UNLOADCODE	39
	DISPLAYSYMBOL	40
	DISPLAYCODE	41
3.2.5	Other commands.....	42
	H	42
Appendix A Configuring the EBC Debugger under EFI Shell		45
A.1	EBC Debugger Configuration.....	45
A.2	Where is EBC Debugger Configuration.....	45
A.3	Command Summary.....	45
A.3.1	Break class commands	45
	BOC 45	
	BOCX 47	
	BOR 48	
	BOE 49	
	BOT 50	
	BOK 51	

Figures

Figure 1. EBC Debug session – step 1	7
Figure 2. EBC Debug session – step 2	8
Figure 3. EBC Debug session – step 3	8

Tables

Table 1. EBC Debugger Commands.....	9
Table 2. EBC Debugger Configuration Commands	45

Revision History

Revision Number	Description	Revision Date
0.1	Initial release.	January 2007
0.2	Draft candidate	February 2007
0.21	Updated the document for the new features: - - Load symbol files from any file systems - - Use the symbol name in BP, G, DB, EB commands - - Add break on keyboard input feature	March 2007

1 *Introduction*

1.1 Overview

This document describes the information on how to use an EBC debugger on EFI implementation. The following chapters include:

How to use an EBC debugger

Description for each EBC debugger command

1.2 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

Component

An executable image. Components defined in this specification support one of the defined module types.

EFI

Generic term that refers to one of the versions of the EFI specification: EFI 1.02, EFI 1.10, UEFI 2.0, UEFI 2.1, or a later UEFI specification.

EFI 1.10 Specification

Intel Corporation published the Extensible Firmware Interface Specification. Intel donated the EFI specification to the Unified EFI Forum, and the UEFI now owns future updates of the EFI specification. See UEFI Specifications.

GUID

Globally Unique Identifier. A 128-bit value used to name entities uniquely. An individual without the help of a centralized authority can generate a unique GUID. This allows the generation of names that will never conflict, even among multiple, unrelated parties.

Module

A module is either an executable image or a library instance. For a list of module types supported by this package, see module type.

UEFI Application

An application that follows the UEFI specification. The only difference between a UEFI application and a UEFI driver is that an application is unloaded from memory when it exits regardless of return status, while a driver that returns a successful return status is not unloaded when its entry point exits.

UEFI Driver

A driver that follows the UEFI specification.

UEFI Specification Version 2.0

First version of the EFI specification released by The Unified EFI Forum. This specification builds on the EFI 1.10 specification and transfers ownership of the EFI specification from Intel to a non-profit, industry trade organization.

UEFI Specification Version 2.1

Current version of the EFI specification released by the Unified EFI Forum.

The Unified EFI Forum

A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see www.uefi.org.

1.3 Conventions used in this document

This document uses the typographic and illustrative conventions described below.

1.3.1 Pseudo-code conventions

Pseudo code is presented to describe algorithms in a more concise form. None of the algorithms in this document are intended to be compiled directly. The code is presented at a level corresponding to the surrounding text.

In describing variables, a list is an unordered collection of homogeneous objects. A queue is an ordered list of homogeneous objects. Unless otherwise noted, the ordering is assumed to be First In First Out (FIFO).

Pseudo code is presented in a C-like format, using C conventions where appropriate. The coding style, particularly the indentation style, is used for readability and does not necessarily comply with an implementation of the Extensible Firmware Interface Specification.

1.3.2 Typographic conventions

This document uses the typographic and illustrative conventions described below:

Plain text	The normal text typeface is used for the vast majority of the descriptive text in a specification.
------------	--

<u>Plain text (blue)</u>	Any plain text that is underlined and in blue indicates an active link to the cross-reference. Click on the word to follow the hyperlink.
Bold	In text, a Bold typeface identifies a processor register name. In other instances, a Bold typeface can be used as a running head within a paragraph.
<i>Italic</i>	In text, an Italic typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.
BOLD Monospace	Computer code, example code segments, and all prototype code segments use a BOLD Monospace typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.
<u>Bo`ld Monospace</u>	Words in a Bold Monospace typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink.
<i>Italic Monospace</i>	In code or in text, words in Italic Monospace indicate placeholder names for variable information that must be supplied (i.e., arguments).
Plain Monospace	In code, words in a Plain Monospace typeface that is a dark red color but is not bold or italicized indicate pseudo code or example code. These code segments typically occur in one or more separate paragraphs.

See the glossary sections in the EFI 1.10 Specification and in the EFI Documentation help system for definitions of terms and abbreviations that are used in this document or that might be useful in understanding the descriptions presented in this document.

See the references sections in the EFI 1.10 Specification and in the in the EFI Documentation help system for a complete list of the additional documents and specifications that are required or suggested for interpreting the information presented in this document:

The EFI 1.10 Specification is available from the EFI web site <http://developer.intel.com/technology/efi/>. The EFI Documentation help system is available from the EFI web site <http://developer.intel.com/technology/efi/help/efidocs.htm>.

1.4 Related Information

The following publications and sources of information may be useful, or are referred to by this document:

Extensible Firmware Interface Specification, Version 1.10, Intel, 2001,
<http://developer.intel.com/technology/efi>.

Unified Extensible Firmware Interface Specification, Version 2.0, Unified EFI, Inc,
2006, <http://www.uefi.org>.

Unified Extensible Firmware Interface Specification, Version 2.1, Unified EFI, Inc,
2007, <http://www.uefi.org>.

2 *Getting Started*

2.1 What is the EBC Debugger?

The EBC Debugger is a tool that can help a user to debug an EBC driver or an EBC application in the EFI shell environment.

The EBC Debugger is an EFI native (service) driver. It is an EBC interpreter with debug ability.

2.2 Where is EBC Debugger

The EBC Debugger is on the CD in the \EbcDebugger\ directory. The included binaries only support Intel® 64, IA-32, and Intel® Itanium® instruction set architectures.

2.3 Prerequisite

When the EBC Debugger is loaded, it will unload the existing EBC interpreter. So the user should ensure that there are no other EBC interpreters loaded after that.

The EBC Debugger uses ConOut and ConIn as input and output interface. ConIn and ConOut are required to operate the EBC debugger.

(Optional) In order to support symbolic debugging, the user needs to use DEBUG version of the driver and provide the .MAP file from the building of the driver. The .MAP file could be put into any file system. The EBC Debugger searches the .MAP file on all file systems automatically.

(Optional) In order to support source level debugging, the user need to use DEBUG version driver and provide both the .MAP file and the .COD files from the building of the driver. The .MAP and .COD files could be put into any file system. The EBC Debugger searches the .MAP and .COD files on all file systems automatically.

If two source files have same name, then they will have COD files with the same name. This is not supported.

2.4 Load the EBC Debugger

As EBC Debugger is a driver, there are 2 ways to load it.

The user can load it manually by using shell command "load EbcDebugger.efi."

The user can build the EbcDebugger.efi to the firmware image, so it will be loaded automatically in system booting.

2.5 Run the EBC Debugger

If the EBC debugger is loaded it will automatically start when it meets one of the following conditions:

An EBC image starts.

Native-to-EBC thunk code is called.

An EBC exception happens. For example, EBC Breakpoint exception.

When the EBC debugger starts, the EBC debugger prompt will be displayed. User can then use EBC debugger commands in this shell-like environment.

2.6 A typical EBC Debug session

an example of a typical EBC debug session follows. EbcTest.efi is an EBC driver, EbcTest.map is the .MAP file, EbcTest.cod, and EbcTestSub.cod are the .COD files. These are located in the CD directory \EbcTest\.

The steps followed by the user are:

- 1) On the target system, boot an EFI system.
- 2) Get the debugger loaded into memory. (see 2.4)
- 3) Copy all the .MAP file and .COD file to 1st file system, for example: fsnt0:\ebctest\
- 4) Load the driver (load ebctest.efi). This causes the EBC debugger prompt to display with the interpreter stopped at the EBC driver's entry point. (see 2.5)
- 5) Load the symbol files at Debugger command prompt.
 - a) type "loadsymbols ebctest\ebctest.map a" (the "a" switch causes the debugger to load all .cod file in same directory See Figure 1.

```

fsmt1:\> load ebctest.efi
EBC Interpreter Version - 1.0
EBC Debugger Version - 0.1
Break on Entrypoint
08D2E620: B7 37 00 00 01
08D2E625: 00          MOVIqd    R7, 65536
08D2E626: 00 06          BREAK     6
08D2E628: 60 00 50 80     MOVqw    R0, R0(-0,-80)
08D2E62C: 77 36 00 00     MOVIqw    R6, 0
08D2E630: B9 37 CA 03 00
08D2E635: 00          MOVreld   R7, 0x000003CA

Please enter command now, 'h' for help.
(Using <Command> -b <...> to enable page break.)

EDB > loadsymbol ebctest\ebctest.map a

EDB > _

```

Figure 1. EBC Debug session – step 1

- 6) The user will list all symbols with the `ln` command and find the address of `EfiMain()` routing (0x8D2E51A in this case) see Figure 2.

```

EDB > ln
Symbol File Name: ebctest.map
  Address  Type  Symbol
  =====
0x08D2E442 (F) TestSubRoutine (EbcTest.obj)
0x08D2E51A (F) EfiMain (EbcTest.obj)
0x08D2E600 (F) TestSubRoutineSub (EbcTestSub.obj)
0x08D2E620 (F) EfiStart (EbcLib:EbcLib.obj)
0x08D2E800 (F) varbss_init_C:\efi_src\TIANO\Edk\Sample\Universal\Ebc\Dxe\EbcT
est\EbcTest$c45b6d8ef (EbcTest.obj)
0x08D2E820 (F) varbss_init_C:\efi_src\TIANO\Edk\Sample\Universal\Ebc\Dxe\EbcT
est\EbcTestSub$c45b6d8ef (EbcTestSub.obj)
0x08D2EA00 (GV) CrtThunkBegin (EbcLib:EbcLib.obj)
0x08D2EA04 (GV) CrtThunkEnd (EbcLib:EbcLib.obj)
0x08D2EA08 (GV) CrtBegin (EbcLib:EbcLib.obj)
0x08D2EA14 (GV) CrtEnd (EbcLib:EbcLib.obj)
0x08D2EC70 (GV) TestStr (EbcTest.obj)
0x08D2EC78 (GV) TestVariable1 (EbcTest.obj)
0x08D2EC80 (GV) TestSubVariableSub (EbcTestSub.obj)
0x08D2E400 (SF) TestSubRoutine2 (EbcTest.obj)

```

Figure 2. EBC Debug session – step 2

- 7) The user allows the program to run freely until the **EfiMain()** routine (use the command "G til 8d2e51a"). The program execution is now at the Image's Entry point. The user can set breakpoints and debug in their code. See Figure 3.

```

EDB > g til 8d2e51a
Break on GoTil
[EfiMain]:
08D2E51A: 60 00 70 80      MOVq    R0, R0(-0,-112)
;117 ; {
08D2E51E: 77 58 58 00 34
08D2E523: 12                MOVI    @R0(+0,+88), 4660
;118 ;   UINT16 test = 0x1234;
08D2E524: 72 87 01 12      MOVnw   R7, @R0(+1,+128)
08D2E528: 72 F7 85 21      MOVnw   R7, @R7(+5,+24)
;121 ;   EFI_STATUS Status;
;121 ;
;121 ;   SystemTable->ConOut->OutputString (
08D2E52C: 72 84 01 12      MOVnw   R4, @R0(+1,+128)
EDB > _

```

Figure 3. EBC Debug session – step 3

The user can also set breakpoints in the source code using the **EFI_BREAKPOINT()** macro, which is defined as **_break(3)** in debug builds. This will result in the EBC debugger stopping at that place in the code.

Please see Chapter 3 for details on the commands of the EBC debugger.

3 *EBC Debugger Command Description*

3.1 Overview

3.1.1 Command Summary

Table 1 lists all EBC debugger commands.

Table 1. EBC Debugger Commands

Class	Command	Description
Execution	G	continue to run the program.
	T	step into.
	P	step over.
	O	step out.
	Q	reset the debugger to default value and go.
Break	BOC	break on CALL.
	BOCX	break on CALLEX.
	BOR	break on RET.
	BOE	break on Driver Entrypoint.
	BOT	break on Native Thunk.
	BOK	break on Keyboard Input
	BL	breakpoint list
	BP	breakpoint set
	BC	breakpoint clear
	BD	breakpoint disable
	BE	breakpoint enable
Information	K	show/clear call-stack
	TRACE	show/clear trace instruction branch
	R	display/modify register
	L	show/load instruction assembly count
	SCOPE	load scope address
	DB, DW, DD, DQ	display memory

	EB, EW, ED, EQ	modify memory
Symbol	LN	list the symbol
	LOADSYMBOL	load the symbol file
	UNLOADSYMBOL	unload the symbol file
	LOADCODE	load the code file
	UNLOADCODE	unload the code file
	DISPLAYSYMBOL	disable/enable the symbol output
	DISPLAYCODE	disable/enable the source code only output
Other	H	help

3.1.2 Explanation of Command Description Layout

The description of each command is composed of four sections: **Summary**, **Usage**, **Function Key**, and **Description**.

Summary is a brief explanation of the function of the command. **Usage** describes how the command is used. **Function Key** is the fast way to run this command. **Description** describes the details of the command.

3.2 EBC Debugger Commands

3.2.1 Execution class commands

G

Summary

continue to run the program.

Usage

```
G [til <Address|Symbol>]

(No Argument) - It means continue run the program.
til           - It means continuing run the program till IP is the
Address.
<Address>     - The hexadecimal address user wants to break at.
<Symbol>      - The symbol name for target address user want to break
at. It has following format [MapFileName:]SymbolName.
```

Function Key

[F5]

Description

Use of the go command causes the debugger not to interrupt execution of the EBC image. The debugger will only break execution of the interpreter if it encounters an exception (including an EBC breakpoint).

Examples

```
Examples:
* To continue run the program:
EDB > G

* To continue run the program until IP is 8D2F51A:
EDB > G TIL 8D2F51A
Break on GoTil
[EfiMain]:
08D2F51A: 60 00 70 80    MOVqw      R0, R0(-0,-122)
;117 ; {
08D2F51E: 77 58 58 00 34
08D2F523: 12                MOVIww      @R0(+0,+88), 4660
;118 ;   UINT16 test = 0x1234;
08D2F524: 72 87 01 12    MOVnw      R7, @R0(+1,+128)
08D2F528: 72 F7 85 21    MOVnw      R7, @R7(+5,+24)
;121 ;   EFI_STATUS Status;
;121 ;
;121 ;   SystemTable->ConOut->OutString (
08D2F52C: 72 84 01 12    MOVnw      R4, @R0(+1,+128)
```

T

Summary

step into.

Usage

```
T
(No Argument)
```

Function Key

```
[F8]
```

Description

The step into command will cause the EBC debugger to step a single instruction. If the instruction is a call to internal code (CALL), then the debugger will break at the new function CALL.

Examples

```
Examples:
* To step into the program:
  EDB > T
```

P

Summary

step over.

Usage

P

(No Argument)

Function Key

[F10]

Description

The step over command causes the EBC debugger to step a single instruction. If the instruction is a call to internal code (CALL), then the external call is made and the debugger breaks at the instruction following the CALL.

Examples

```
Examples:
* To step over the program:
  EDB > P
```

O

Summary

step out.

Usage

```
O  
  
(No Argument)
```

Function Key

```
[F11]
```

Description

The step out command causes the EBC debugger to step out function calls. The function executes, but the debugger stops after the called function returns.

Examples

```
Examples:  
* To step out the program:  
  EDB > O
```

Q

Summary

reset the debugger to default value and go.

Usage

Q

(No Argument)

Function Key

(None)

Description

The quit command will reset the debugger to default value and go.

Examples

Examples:

```
* To reset the debugger to default value and go:  
EDB > Q
```

3.2.2 Break class commands

BOC

Summary

break on CALL.

Usage

```
BOC [on|off]

    (No Argument) - show current state
    on             - enable break-on-call
    off            - disable break-on-call
```

Function Key

(None)

Description

Enabling break-on-call will cause the debugger to halt execution and display the debugger prompt prior to executing any EBC CALL (to EBC) instructions.

Examples

```
Examples:
* To enable break-on-CALL:
  EDB > BOC ON

* To show the current state:
  EDB > BOC
  BOC ON
```

BOCX

Summary

break on CALLEX.

Usage

`BOCX [on|off]`

(No Argument)	- show current state
on	- enable break-on-callex
off	- disable break-on-callex

Function Key

(None)

Description

Enabling break-on-callex causes the debugger to halt execution and display the debugger prompt prior to executing EBC CALLEX (think out) instructions.

Examples

```
Examples:
* To enable break-on-CALLEX:
  EDB > BOCX ON

* To show the current state:
  EDB > BOCX
  BOCX ON
```

BOR

Summary

break on RET.

Usage

`BOR [on|off]`

(No Argument)	- show current state
on	- enable break-on-return
off	- disable break-on-return

Function Key

(None)

Description

Enabling break-on-return will cause the debugger to halt execution and display the debugger prompt prior to executing EBC RET instructions.

Examples

```
Examples:
* To enable break-on-RET:
  EDB > BOR ON

* To show the current state:
  EDB > BOR
  BOR ON
```

BOE

Summary

break on Driver Entrypoint.

Usage

```
BOE [on|off]
```

```
(No Argument) - show current state
on             - enable break-on-entrypoint
off            - disable break-on-entrypoint
```

Function Key

(None)

Description

Enabling break-on-entrypoint causes the debugger to halt execution and display the debugger prompt prior to start a driver entry point. (Default is on).

Examples

```
Examples:
* To disable break-on-entrypoint:
  EDB > BOE OFF

* To show the current state:
  EDB > BOE
  BOE OFF
```

BOT

Summary

break on Native Thunk.

Usage

```
BOT [on|off]

    (No Argument) - show current state
    on             - enable break-on-thunk
    off            - disable break-on-thunk
```

Function Key

(None)

Description

Enabling break-on-thunk will cause the debugger to halt execution and display the debugger prompt prior to start native call EBC thunk. (Default is on)

Examples

```
Examples:
* To enable break-on-thunk:
  EDB > BOT ON

* To show the current state:
  EDB > BOT
  BOT ON
```

BOK

Summary

break on Keyboard Input.

Usage

```
BOK [on|off]
```

```
(No Argument) - show current state
on             - enable break-on-key
off            - disable break-on-key
```

Function Key

(None)

Description

Enabling break-on-key will cause the debugger to halt execution and display the debugger prompt after press any key. It is useful when an EBC driver has an infinite loop.

Examples

```
Examples:
* To enable break-on-key:
  EDB > BOK ON

* To show the current state:
  EDB > BOK
  BOK ON
```

BL

Summary

breakpoint list.

Usage

BL

(No Argument) - show the state for current breakpoint

Function Key

(None)

Description

List Breakpoint

Examples

```

Examples:
* To list breakpoint:
  EDB > BL
Breakpoint:
  Index   Address                               Status
=====
  0       0x00000000008D2F52C                  *
```

BP

Summary

breakpoint set.

Usage

```
BP <Address|Symbol>
```

<Address> - Hexical breakpoint address

<Symbol> - Symbol name for breakpoint address. It has following format
[MapFileName:]SymbolName.

Function Key

(None)

Description

Set Breakpoint

Examples

```
Examples:  
* To set breakpoint:  
EDB > BP 8D2E52C
```

BC

Summary

breakpoint clear.

Usage

```
BC <Index>|*
```

```
<Index>  - Decimal breakpoint index, which can be got from BL command  
*        - For all the breakpoint
```

Function Key

(None)

Description

Clear Breakpoint

Examples

```
Examples:  
* To clear breakpoint:  
EDB > BC 0
```

BD

Summary

breakpoint disable.

Usage

```
BD <Index>|*
```

```
<Index>  - Decimal breakpoint index, which can be got from BL command  
*        - For all the breakpoint
```

Function Key

(None)

Description

Disable Breakpoint

Examples

```
Examples:  
* To disable breakpoint:  
EDB > BD 0
```

BE

Summary

breakpoint enable.

Usage

```
BE <Index>|*
```

```
<Index>  - Decimal breakpoint index, which can be got from BL command  
*        - For all the breakpoint
```

Function Key

(None)

Description

Enable Breakpoint

Examples

```
Examples:  
* To enable breakpoint:  
EDB > BE 0
```

3.2.3 Information class commands

K

Summary

show/clear call-stack.

Usage

```
K [p [<ParameterNum>] | c]

      (No Argument) - Show current call-stack
      p             - Show current call-stack with parameters
      ParameterNum - Decimal call-stack parameters number, 8 by default, 16
as max
      c             - Clear current call-stack
```

Function Key

(None)

Description

The call-stack command will show or clear the current call-stack.

Examples

```
Examples:
* To show the current call-stack:
  EDB > K
Call-Stack (TOP):
      Caller              Callee              Name
      =====
0x0000000008D2F55A 0x0000000008D2F600 TestSubRoutineSub()
0x0000000008D2F750 0x0000000008D2F51A EfiMain()
0x00000000FFFFFFFF 0x0000000008D2F620 EfiStart()

* To show the current call-stack with parameter:
  EDB > K P 2
Call-Stack (TOP):
      Caller              Callee              Name
      =====
0x0000000008D2F55A 0x0000000008D2F600 TestSubRoutineSub()
      Parameter Address (0x08B26F24) (
        0x000000001, 0x000000005
      )
0x0000000008D2F750 0x0000000008D2F51A EfiMain()
      Parameter Address (0x08B26FA4) (
        0x08D2D710, 0x04C6FE90
      )
0x00000000FFFFFFFF 0x0000000008D2F620 EfiStart()
      Parameter Address (0x08B26FF4) (
        0xAFAFAFAF, 0xAFAFAFAF
      )
```

TRACE

Summary

show/clear trace instruction branch.

Usage

TRACE [c]

(No Argument) - Show current instruction branch
c - Clear current instruction branch

Function Key

(None)

Description

The trace command will show or clear the latest instruction branch.

Examples

Examples:

* To show the current instruction branch:
EDB > TRACE

Instruction Trace (->Latest):

Source Addr	Destination Addr	Type
0x0000000008D2F652	0x0000000008D2F6CE	(JMP8)
0x0000000008D2F6E8	0x0000000008D2F6EA	(JMP8)
0x0000000008D2F702	0x0000000008D2F704	(JMP8)
0x0000000008D2F70C	0x0000000008D2F72A	(JMP8)
0x0000000008D2F744	0x0000000008D2F704	(JMP8)
0x0000000008D2F70C	0x0000000008D2F70E	(JMP8)
0x0000000008D2F728	0x0000000008D2F800	(CALL)

R

Summary

display/modify register.

Usage

```
R [<Register> <Value>]

    (No Argument) - Display all registers
    <Register>    - EBC VM register name (R0~R7, Flags, ControlFlags, and
IP
    <Value>       - The Hexical value of register
```

Function Key

[F2]

Description

The register command is used to display or modify the contents of EBC VM registers. (R0~R7, Flags, IP).

Examples

```
Examples:
* To show the current register:
  EDB > R
R0 - 0x000000000008b26F14, R1 - 0x00000000000000
R2 - 0x000000000008b26F14, R3 - 0x00000000000000
R4 - 0x000000000008b26F14, R5 - 0x00000000000000
R6 - 0x000000000008b26F14, R7 - 0x00000000000000
Flags - 0x0000000000000001, ControlFlags - 0x0000000000000000
Ip - 0x000000000008D2F61A

* To update the current register:
  EDB > R R1 1
```

L

Summary

show/load instruction assembly count.

Usage

```
L [<Count>]
```

```
(No Argument) - List current assembly code  
<Count>       - The decimal instruction assembly count
```

Function Key

[F4]

Description

The list assembly command will disassemble instructions starting with the current EBC VM instruction pointer. (by default 5 instructions).

Examples

```
Examples:  
* To show the current assembly:  
EDB > L
```

SCOPE

Summary

load scope address.

Usage

```
SCOPE <Address|Symbol>
```

<Address> - The Hexical address where user wants to see the assembly code

<Symbol> - Symbol name for scope address. It has following format
[MapFileName:]SymbolName.

Function Key

(None)

Description

The list assembly command will disassemble instructions starting with the current EBC VM instruction pointer. (by default 5 instructions).

Examples

Examples:

- * To load the scope address:
EDB > SCOPE 8D2F61A

DB, DW, DD, DQ

Summary

display memory.

Usage

```
D[B|W|D|Q] <Address|Symbol> [<Count>]
```

```
<Address> - The hexical memory address
<Symbol> - Symbol name for memory address. It has following format
[MapFileName:]SymbolName.
<Count> - The hexical memory count (not set means 1)
```

Generally only the global C variables are stored in .MAP file. For the local variables or static variables, user still needs to access them from memory address.

Function Key

(None)

Description

Display BYTES/WORDS/DWORDS/QWORDS Memory.

Examples

```
Examples:
* To show the memory:
EDB > DD 8D2E000 8
08D2E000: 30726670 00000000 08DAAA1C 08D2E088
08D2E010: AFAFAFAF AFAFAFAF AFAFAFAF AFAFAFAF
```

EB, EW, ED, EQ

Summary

modify memory.

Usage

```
E[B|W|D|Q] <Address|Symbol> <Value>
```

```
<Address> - The hexical memory address
<Symbol>  - Symbol name for memory address. It has following format
[MapFileName:]SymbolName.
<Value>   - The hexical memory value
```

Note: Generally only the global C variables are stored in .MAP file. For the local variables or static variables, user still needs to access them from memory address.

Function Key

(None)

Description

Enter BYTES/WORDS/DWORDS/QWORDS Memory.

Examples

```
Examples:
* To modify the memory:
  EDB > ED 8D2FC78 8
```

3.2.4 Symbol class commands

LN

Summary

list the symbol.

Usage

```
LN [[F <SymbolFile>] [S <Symbol>]] | <Address>
```

```
(No Argument) - List all the symbol
F <SymbolFile> - List the symbol in this symbol file only
S <Symbol>      - List this symbol only
<Address>      - The hexical memory address, which user want to find
the symbol for.
```

Function Key

(None)

Description

The show symbol command will list all the current symbol. It can list the symbol in one symbol file, or list the same symbol in all the files. It can also list the symbol according to nearest address. (In the result - type field, F means Function, SF means Static Function, GV means Global Variable)

Examples

```
Examples:
* To list the symbol:
  EDB > LN
Symbol File Name: ebctest.map
  Address      Type      Symbol
  =====
0x08D2F442 ( F) TestSubRoutine (EbcTest.obj)
0x08D2F51A ( F) EfiMain (EbcTest.obj)
0x08D2F600 ( F) TestSubRoutineSub (EbcTest.obj)
0x08D2F620 ( F) EfiStart (EbcLib:EbcLib.obj)
0x08D2F800 ( F)
varbss_init C:\efi_src\TIANO\Edk\Sample\Universal\Ebc\Dxe\EbcTest\EbcTest
$c45b6d8ef (EbcTest.obj)
0x08D2F820 ( F)
varbss_init C:\efi_src\TIANO\Edk\Sample\Universal\Ebc\Dxe\EbcTest\EbcTest
Sub$c45b6d8ef (EbcTestSub.obj)
0x08D2FA00 (GV) CrtThunkBegin (EbcLib:EbcLib.obj)
0x08D2FA04 (GV) CrtThunkEnd (EbcLib:EbcLib.obj)
0x08D2FA08 (GV) CrtBegin (EbcLib:EbcLib.obj)
0x08D2FA14 (GV) CrtEnd (EbcLib:EbcLib.obj)
0x08D2FC70 (GV) TestStr (EbcTest.obj)
0x08D2FC78 (GV) TestVariable1 (EbcTest.obj)
0x08D2FC80 (GV) TestSubVariableSub (EbcTestSub.obj)
0x08D2F400 (SF) TestSubRoutine2 (EbcTest.obj)

* To list the nearest symbol:
EDB > LN 8d2f500
Symbol at Address not found, print nearest one!
Symbol File Name: ebctest.map
  Address      Type      Symbol
  =====
0x08D2F51A ( F) EfiMain

* To list the symbol with name:
EDB > LN S EfiMain
Symbol File Name: ebctest.map
  Address      Type      Symbol
  =====
0x08D2F51A ( F) EfiMain (EbcTest.obj)
```

LOADSYMBOL

Summary

load the symbol file.

Usage

```
LOADSYMBOL <SymbolFile> [a]
```

```
SymbolFile - The EBC symbol file (Its name should be XXX.MAP)  
a          - Automatically load code files in the same dir
```

Function Key

(None)

Description

The load symbol command will load the ebc map file. Then it parses the function name and global variable, and the print real name when do the disassembly. (Symbol file name should be XXX.MAP).

Examples

Examples:

* To load the symbol:

```
EDB > LOADSYMBOL ebctest\ebctest.map
```

* To load the symbol and related code:

```
EDB > LOADSYMBOL ebctest\ebctest.map a
```

UNLOADSYMBOL

Summary

unload the symbol file.

Usage

```
UNLOADSYMBOL <SymbolFile> [a]
```

SymbolFile - The EBC symbol file (Its name should be XXX.MAP)

Function Key

(None)

Description

The unload symbol command will unload the ebc map and cod file. After that the name will not be print.

Examples

```
Examples:  
* To unload the symbol:  
EDB > UNLOADSYMBOL ebctest.map
```

LOADCODE

Summary

load the code file.

Usage

```
LOADCODE <CodeFile> <SymbolFile>
```

```
CodeFile    - The EBC code file (Its name should be XXX.COD)
SymbolFile  - The EBC symbol file (Its name should be XXX.MAP)
```

Function Key

(None)

Description

The load code command will load the ebc cod file. Then it parses the cod file, and the print source code when do the disassembly. (Code file name should be XXX.COD).

Examples

```
Examples:
* To load the code:
  EDB > LOADCODE ebctest\ebctest.cod ebctest.map
```

UNLOADCODE

Summary

unload the code file.

Usage

```
UNLOADCODE <CodeFile> <SymbolFile>
```

```
CodeFile    - The EBC code file (Its name should be XXX.COD)
SymbolFile  - The EBC symbol file (Its name should be XXX.MAP)
```

Function Key

(None)

Description

The unload code command will unload the ebc cod file. After that the source code will not be print.

Examples

```
Examples:
* To unload the code:
  EDB > UNLOADCODE ebctest\ebctest.cod ebctest.map
```

DISPLAYSYMBOL

Summary

disable/enable the symbol output.

Usage

```
DISPLAYSYMBOL [on|off]
```

```
(No Argument) - swtich symbol output state to another one  
on             - enable symbol output  
off            - disable symbol output
```

Function Key

```
[F3]
```

Description

The display symbol command will configure the symbol show or not-show when disassembly.

Examples

```
Examples:  
* To siwtch display symbol:  
EDB > DISPLAYSYMBOL
```

DISPLAYCODE

Summary

disable/enable the source code only output.

Usage

```
DISPLAYCODE [on|off]
```

```
(No Argument) - swtich source only output state to another one  
on             - enable source only output  
off            - disable source only output
```

Function Key

[F6]

Description

The display code command will configure the source code only show or miscellaneous source code with assembly.

Examples

```
Examples:  
* To siwtch display code:  
EDB > DISPLAYCODE
```

3.2.5 Other commands

H

Summary

Help.

Usage

```
H [<Command>]
```

```
(No Argument) - show help information for all command  
Command       - show detail help information for this command
```

Function Key

```
[F1]
```

Description

The help command will print help information for each command.

Examples

Examples:

- To print help:

```
EDB > H
Execution:
  G/[F5] - continue to run the program
  T/[F8] - step into
  P/[F10] - step over
  O/[F11] - step out
  Q - reset the debugger to default value and go
Break:
  BO[C|CX|R|E|T|K] - break on CALL/CALLEX/RET/Entrypoint/Native
Thunk/Key
  B[L|P|C|D|E] - breakpoint list/set/clear/disable/enable
Information:
  K - show/clear call-stack
  TRACE - show/clear trace instruction branch
  R/[F2] - display/modify register
  L/[F4] - show/load instruction assembly count
  SCOPE - load scope address
  [D|E] [B|W|D|Q] - display/modify memory
Symbol:
  LN - list the symbol
  [UN]LOADSYMBOL - load/unload the symbol file
  [UN]LOADCODE - load/unload the code file
  DISPLAYSYMBOL/[F3] - disable/enable the symbol output
  DISPLAYCODE/[F6] - disable/enable the source code only output
```

EDB > H G

The go command is used to cause the debugger to not interrupt execution of the EBC image. The debugger will only break execution of the interpreter if an exception is encountered (including an EBC breakpoint).

```
G [til <Address|Symbol>]
  (No Argument) - It means continue run the program.
  til - It means continuing run the program till IP is the
Address.
  <Address> - The hexical address user want to break at.
  <Symbol> - The symbol name for target address user want to break
at. It has following format [MapFileName:]SymbolName.
```


Appendix A

Configuring the EBC Debugger under EFI Shell

A.1 EBC Debugger Configuration

Sometimes the user may want to disable all Break conditions and just let the EBC image run. How can this be done and then reversed at the user's discretion. The EFI shell application EbcDebuggerConfig accomplishes this.

A.2 Where is EBC Debugger Configuration

EBC Debugger Configuration is on the CD in \EbcDebuggerConfig\ directory. The binaries only support Intel® 64, IA-32, and Intel® Itanium® architectures.

A.3 Command Summary

Table 2 lists all EBC debugger configuration commands.

Table 2. EBC Debugger Configuration Commands

Class	Command	Description
Break	BOC	break on CALL.
	BOCX	break on CALLEX.
	BOR	break on RET.
	BOE	break on Driver Entrypoint.
	BOT	break on Native Thunk.
	BOK	break on Keyboard Input

A.3.1 Break class commands

BOC

Summary

break on CALL.

Usage

```
BOC [on|off]

      (No Argument) - show current state
      on             - enable break-on-call
      off            - disable break-on-call
```

Description

Enabling break-on-call will cause the debugger to halt execution and display the debugger prompt prior to executing any EBC CALL (to EBC) instructions.

Examples

```
Examples:
* To enable break-on-CALL:
  Shell> EDBCFCFG BOC ON

* To show the current state:
  Shell> EDBCFCFG BOC
  BOC ON
```

BOCX

Summary

break on CALLEX.

Usage

```
BOCX [on|off]
```

```
(No Argument) - show current state
on             - enable break-on-callex
off            - disable break-on-callex
```

Description

Enabling break-on-callex will cause the debugger to halt execution and display the debugger prompt prior to executing EBC CALLEX (think out) instructions.

Examples

```
Examples:
* To enable break-on-CALLEX:
  Shell> EDBCFCG BOCX ON

* To show the current state:
Shell> EDBCFCG BOCX
BOCX ON
```

BOR

Summary

break on RET.

Usage

`BOR [on|off]`

(No Argument)	- show current state
on	- enable break-on-return
off	- disable break-on-return

Description

Enabling break-on-return will cause the debugger to halt execution and display the debugger prompt prior to executing EBC RET instructions.

Examples

```
Examples:
* To enable break-on-RET:
  Shell> EDBCFG BOR ON

* To show the current state:
Shell> EDBCFG BOR
BOR ON
```

BOE

Summary

break on Driver Entrypoint.

Usage

```
BOE [on|off]

    (No Argument) - show current state
    on             - enable break-on-entrypoint
    off            - disable break-on-entrypoint
```

Description

Enabling break-on-entrypoint will cause the debugger to halt execution and display the debugger prompt prior to start a driver entry point. (Default is on).

Examples

```
Examples:
* To disable break-on-entrypoint:
  Shell> EDBCFG BOE OFF

* To show the current state:
Shell> EDBCFG BOE
BOE OFF
```

BOT

Summary

break on Native Thunk.

Usage

```
BOT [on|off]

    (No Argument) - show current state
    on             - enable break-on-thunk
    off            - disable break-on-thunk
```

Description

Enabling break-on-thunk will cause the debugger to halt execution and display the debugger prompt prior to start native call EBC thunk. (Default is on)

Examples

```
Examples:
* To enable break-on-thunk:
  Shell> EDBCFCG BOT ON

* To show the current state:
Shell> EDBCFCG BOT
BOT ON
```

BOK

Summary

break on Keyboard Input.

Usage

```
BOK [on|off]
```

```
(No Argument) - show current state
on             - enable break-on-key
off            - disable break-on-key
```

Description

Enabling break-on-key will cause the debugger to halt execution and display the debugger prompt after press any key. It is useful when an EBC driver has an infinite loop.

Examples

```
Examples:
* To enable break-on-key:
  Shell> EDBCFG BOK ON

* To show the current state:
Shell> EDBCFG BOK
BOK ON
```